Do Algorithm Animations Aid Learning?

Michael D. Byrne[1] Richard Catrambone[1] John T. Stasko[2]

[1]School of Psychology
Georgia Institute of Technology
Atlanta, GA 30332-0170

[2]College of Computing
Georgia Institute of Technology
Atlanta, GA 30332-0280

Technical Report GIT-GVU-96-18
August 1996

Abstract

Two experiments examined the general claim that animations can help students learn algorithms more effectively. Animations and instructions that explicitly required learners to predict the behavior of an algorithm were used during training. Post-test problems were designed to measure how well learners could predict algorithm behavior in new situations as well as measure learners' conceptual understanding of the algorithms. In Experiment 1, we found that when learners both viewed an animation and made predictions their performance on novel problems improved compared to a control group's, but the effects of the two manipulations were not distinguishable. In Experiment 2, no effect was found for conceptual measures of learning, but a marginally reliable effect similar to the one seen in Experiment 1 was found for procedural problems. The results from the two experiments suggest that the benefits of animations are not obvious and that in order to determine whether animations can truly aid understanding, teachers and researchers should consider a careful task analysis ahead of time to determine the specific pieces of knowledge that an animation can help a learner acquire and/or practice.

1. Introduction

Educators constantly are seeking new ways to improve instruction, to facilitate learning, and to hold the attention of their students. The power of computers to store vast quantities of information and to simulate environments and conditions that would otherwise be unavailable makes them an intriguing possibility as an instructional aid.

Our work examines whether multimedia capabilities, particularly animation, can facilitate learning. The use of pictures and visualizations as educational aids is accepted practice; textbooks are filled with pictures, and instructors often diagram concepts on the blackboard to assist an explanation.

Animation goes one step further. While static visualizations can provide people with the essence of how something looks, is laid out, or is constituted, animation appears better able to explain a dynamic, evolving process. Cinema has used animation to tell a story for over half a century, but computers now are making animations much easier to design and produce. Computer scientists use animation to help viewers track patterns and observe relationships (Robertson, Card, & Mackinlay, 1993). Animation has been used to explain principles and laws of motion (Rieber, Boyce, & Assad, 1990), to provide on-line help with interface tasks (Sukaviriya, 1990), and to help users follow interface operations (Baecker & Small, 1990).

Formal studies about the effect of animation on understanding have provided mixed results. Rieber and his colleagues found that the addition of animation to help teach science concepts such as Newtonian mechanics had no effect on learning as measured by a multiple choice post-test (Rieber, 1990; Rieber et al., 1990). Palmiter (1993) studied the use of animation to aid computer authoring tasks. Animation initially assisted both accuracy and speed, but after one week had elapsed, the subjects exposed to animations actually had fallen behind the non-animation subjects. Studies by Mayer and Anderson found that animation did help college students learn about mechanical concepts as measured by a creative problem solving post-test (Mayer & Anderson, 1991, 1992).

Our focus is the use of animation to teach programming and to teach students how algorithms work. This is called algorithm animation (Brown, 1988a), and is one particular instance of software visualization (Price, Baecker, & Small, 1993) which is the use of images, graphics, and animation to illustrate computer algorithms, programs, and processes. The video *Sorting Out Sorting*, presented at SIGGRAPH '81, is generally credited with initiating the field of algorithm animation. It showed views of data being sorted by different algorithms to help students understand how the algorithms work and how they compare. Since then, many algorithm animation systems have been created (Brown, 1988b, 1991; Gloor, 1992; Stasko, 1990) and a number are publicly available. Many universities use the systems to help teach algorithms to students. This type of instruction is facilitated by the availability of an "electronic" classroom to

conduct lectures and by the availability of powerful computers with multimedia capabilities for students to work out of the classroom.

An algorithm animation actually serves two fundamental purposes as an instructional aid: it provides a concrete depiction of the abstractions and operations inherent in an algorithm or program, and it portrays the dynamics of a time-evolving process. For these reasons, educators and algorithm animation system developers have hypothesized that algorithm animations can become valuable instructional aids. Unfortunately, this hypothesis is largely based on intuition and anecdotal evidence.

Nevertheless, many computer scientists believe that algorithm animations should assist instruction. This accounts for systems' popularity despite the lack of solid empirical evidence supporting the claim that animations aid understanding.

1.1 What Can Animations Help Students Learn?

Earlier experimental studies of the effect of algorithm animation on student understanding have shown small benefits, but not at the level hoped for by system developers. Stasko, Badre, and Lewis (1993) studied the addition of algorithm animation to traditional textual materials for graduate computer science students learning about pairing heap data structures. They found small, non-significant post-test benefits in the participants seeing the animations. Lawrence, Badre, and Stasko (1994) utilized animation to help teach undergraduates about minimum spanning tree algorithms. They found a significant benefit when students were able to interact with the algorithm animations in a post-class laboratory setting.

In general, the information to be conveyed by an algorithm animation can be broken into two broad areas: procedural and conceptual. Animations might be effective in conveying both types of information. Procedural knowledge represents the sequence of steps carried out by the algorithm. These steps can frequently be displayed explicitly in an animation, thus presumably helping the student to learn and remember them. Conceptual knowledge refers to an understanding of the "meaning" and implications of the algorithm. For instance, conceptual knowledge of a particular algorithm might allow a learner to estimate the efficiency of the algorithm's performance in certain situations without having to mentally "run" the algorithm.

Animations might encourage and make it relatively easy for the learner to make and test predictions of what is going to happen at each step of the algorithm. This prediction element could help the learner understand the algorithm better than a learner who passively takes in the information. A learner could certainly make predictions from a static textual/graphic presentation of an algorithm, but perhaps the advantage of an animation is that it will spontaneously encourage a learner to make the predictions without being prompted and will provide the learner with rapid feedback about the accuracy of his or her predictions. In addition, an animation might be more

salient than a set of static images and thus, might help the student remember the algorithm better. An animation also might help learning because it displays the features that one should presumably attend to. That is, the items in the animation might, by their presence, provide information to the learner that might not be easily discernable otherwise. Finally, an animation might also encourage the learner to self-explain the behavior of the algorithm. Self-explanation can increase the likelihood of a learner to integrate the new information with existing knowledge structures, thus making the learner more likely to transfer the information to novel situations (Catrambone, 1996; Chi, Bassok, Lewis, Reimann, & Glaser, 1989; Chi, De Leeuw, Mei-Hung, & Levancher, 1994).

1.2 Understanding An Algorithm

In order to examine the effects that animations might have on helping learners to understand an algorithm, we have to define what we mean by "understand." This is a difficult question because understanding can be measured in many ways and there may be different types of understanding. For instance, if a student is able to solve a problem in mechanics involving a block sliding down an inclined plane, does that mean he or she understands the relevant part of mechanics or does it mean the student was able to memorize a solution procedure that was demonstrated on a similar example in class? Perhaps the student would be considered to understand mechanics if he or she was able to solve a novel problem, that is, one that could not be solved using the exact same steps as those demonstrated on in-class examples. However, even if the student could solve the novel problem, that could be interpreted as meaning that the student has generalized a solution procedure but still might not truly understand the relationship among force, mass, and acceleration. Perhaps true or deep understanding might be demonstrated only if the learner can, without prompting, apply the knowledge to situations that are far removed from the training cases.

We do not propose to offer a solution to what it means to understand an algorithm. We mention the above to indicate that the measures we use for understanding are necessarily narrow in order to allow us to progress with the research. We measure understanding in two ways. The first is a procedural understanding of an algorithm, that is, being able to determine the step to step behavior of an algorithm in a novel situation. The second is a conceptual understanding, that is, to answer certain questions about the algorithm that involve more than applying the algorithm to a particular data set. We have focused more on procedural knowledge acquisition (although Experiment 2 includes conceptual measures). The primary reason for this is that an animation appears to be particularly well-suited for explicitly conveying the steps of the algorithm and because it is more straightforward experimentally to measure procedural knowledge. In addition, although we did not make use of it in the present work, the procedural knowledge the learner

might acquire about the algorithm could be captured nicely in a production rule framework which can be used as a tool for designing and refining future animations.

Thus, our operational definitions of understanding include a learner's ability to predict algorithm performance on novel problems and to answer certain conceptual questions. Other measures of understanding we considered, but did not use in the present experiments, are to have the learner attempt to implement the algorithm, and to examine how well a student can learn a new algorithm as a function of having studied a prior algorithm.

Given this operationalization of understanding, it is reasonable to ask how learners acquire such an understanding and how animations might aid the acquisition process. As previously discussed, there are a number of ways that animations might help learners. Because feedback is a critical element in various psychological theories of how people acquire procedural knowledge (e.g. Anderson, 1993), we decided to concentrate on the "make and test predictions" aspect of algorithm animation. Both of the experiments presented here manipulate whether learners see animations and whether they are asked to make explicit predictions about algorithm behavior (and receive feedback from their predictions). If animations provide advantages to a learner above and beyond those conferred by simply testing predictions, learners exposed to the animations should outperform those making explicit predictions without seeing animations. On the other hand, if animations simply encourage learners to make predictions (either explicitly or implicitly) and to examine the accuracy of their predictions, then learners in the "making predictions" conditions of the experiments should perform as well as learners who view the animations.

1.3 Qualities of Animations

What qualities should an algorithm animation possess? One possibility is consistency with presentations found in textbooks. This approach would at least provide some ecological validity if we were to find an effect of animations on learning. Another issue is how much does the learner need to know to "appreciate" an animation? That is, perhaps a learner must have some familiarity with the algorithm in order to appreciate how the animation works. Animations might make more of a difference in cases where it would be difficult to demonstrate an algorithm's behavior "statically" such as when showing it on a blackboard. For instance, the resulting reorganization process involving a heap or tree when a new node is added can sometimes be quite cumbersome to show on a blackboard, but can be demonstrated easily in an animation.

Because of our desire to make our initial experiments have some ecological validity to classroom settings, we presented the animations in the context of a larger learning situation. Students in our studies watched a short video-lecture on the algorithm and also read a short text on the algorithm before watching an animation. The animations were designed to have some fidelity to the textbook presentations, again to make them consistent with present classroom learning.

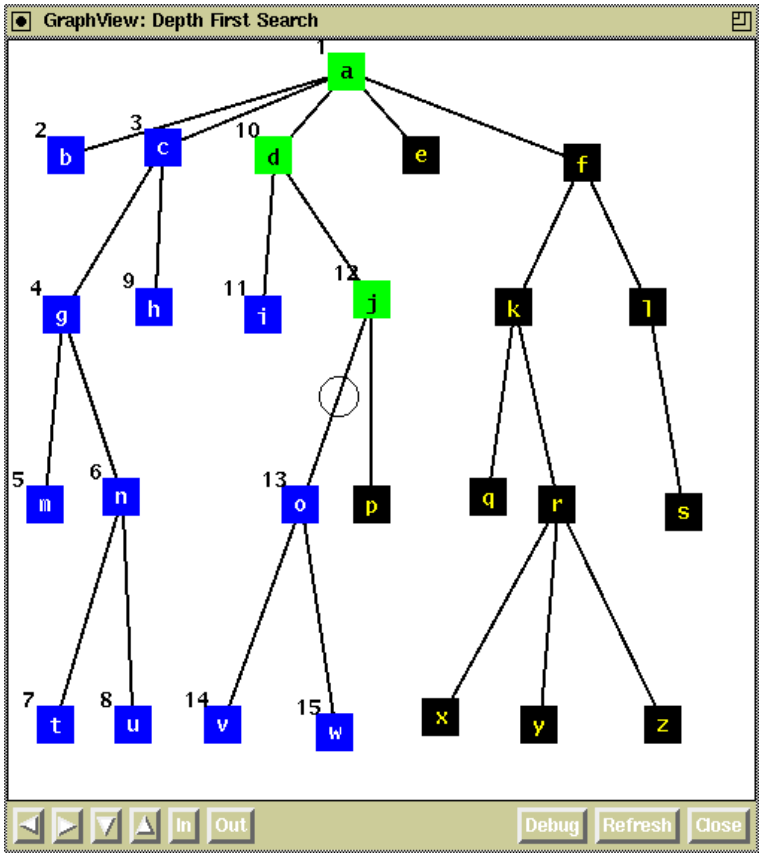
2. Experiment 1

Experiment 1 involved students learning about depth-first search. In a depth-first search, a graph is searched from vertex to vertex by exploring a path from a given vertex as far as that path will go. The path is then backtracked until a vertex is reached that has a path (or "edge") going from it that has not yet been explored. By convention, the vertices are assigned letters and the search starts at the vertex named with the first letter occurring alphabetically. For instance, in Figure 1 the search would start at vertex "a." If more than one edge can be followed from this vertex, the edge connected to the vertex with the lowest letter is chosen first. In Figure 1, the possible choices would be to go to vertices "b," "c," "d," "e," or "f." The edge to vertex "b" is chosen since "b" is the lowest letter of the possible choices. When that vertex is reached, the search continues to the next unvisited vertex connected to the just-reached vertex. This process continues until a vertex is reached that contains no unexplored edges. At this point, the search backs up to the "parent" of the present vertex and an unexplored edge (if there is one) from that vertex is taken. In Figure 1 this means that after vertex "b" is visited, the search would back up to vertex "a" and then one of the remaining unexplored vertices connected to "a" would be visited (i.e., one of the edges leading to vertex "c," "d," "e," or "f" would be taken; the edge to vertex "c" would be chosen since "c" is the lowest letter of the possible choices). If no unexplored edges exist, the search backs up again. This process continues until all vertices have been visited and the search has backed up to the starting vertex.[1]

The animation of the depth-first search algorithm was built with the Polka animation system (Stasko & Kraemer, 1993) and utilizes the traditional view of a graph as a set of vertices and edges. A frame from an animation on an example graph is shown in Figure 1. The vertices are represented by black squares and identified by a letter label. The edges are simply lines drawn between the vertices. The animation utilizes a moving circle (seen between vertices J and O in Figure 1) to represent the progression of the search. The circle moves throughout the graph incrementally and smoothly, traversing edges and encountering vertices. When the search is moving to encounter a new vertex, the circle is solid black. When the search is backtracking from a vertex, the circle is a black outline. When a vertex is first discovered in the search, the vertex becomes solid green and a number representing the order of discovery is placed beside the vertex. When the search moves back past a vertex (i.e., all of its descendants in the depth-first search tree have been found) then the vertex changes to solid blue.

[1]The order of vertices visited in Figure 1 (including all backtracking) are: a, b, a, c, g, m, g, n, t, n, u, n, g, c, h, c, a, d, i, d, j, o, v, o, w, o, j, p, j, d, a, e, a, f, k, q, r, x, r, y, r, z, r, k, f, l, s, l, f, a

Figure 1: Screen Shot of Animation for Depth First Search (DFS)



2.1 Method

Participants. Participants were 88 undergraduate, non-computer science majors at the Georgia Institute of Technology who participated for extra credit in a psychology course. Participants were screened to ensure they had no previous experience with the depth-first search algorithm. The participants were generally high-ability students with a mean self-reported SAT-Quantitative score of 645 and SAT-Verbal score of 535.

Materials and Procedure. Participants first filled out the consent form and a screening questionnaire that included self-reports of SAT scores and grade-point average. Participants were then shown a six-minute videotaped lecture given by one of the authors (JS) that covered depth-first search. After watching the videotape, participants were given a three-page text describing the depth-first search algorithm. This text was adapted from the chapter covering depth-first search in a standard algorithms textbook (Cormen, Leiserson, & Rivest, 1990). Participants in the no-animation/no-prediction condition were given 10 minutes to read the text, all others were given five minutes. Those who received only five minutes with the text then either watched the algorithm

animation (animation/no-prediction condition), made explicit predictions while watching the animation (animation/prediction condition), or made explicit predictions from printed graphs (no-animation/prediction condition). All participants then received the post-test (on paper), on which they had unlimited time to work.

Participants in the no-animation/prediction condition were given two graphs on paper and asked to predict the order in which the vertices would be visited. If they made an error on any prediction, the error was immediately corrected by the experimenter. The other half of the participants who made explicit predictions made those predictions about the animation. The animation was halted immediately after a vertex was visited and the participant asked to state which vertex they thought would be visited next. Errors made by these participants were not pointed out by the experimenter since the animation displayed the correct answer.

The primary dependent variable in this experiment was the score the participants earned on a post-test. Questions on the post-test were divided into two categories: difficult and easy. Easy questions were those that required the participant to determine a single next step of a search or that required determination of a complete search on a graph isomorphic to an example participants had already seen. Difficult questions were those that involved complete searches of novel graphs. Questions on the post-test were scored stringently as being either completely correct or incorrect. In addition, the number of errors made by participants in the prediction condition were also recorded.

The videotaped lecture was displayed on a 19" color television and participants were free to adjust the sound volume to suit their preference. The animations were presented on a Sun SPARCStation2 with a 19" color monitor at a preset speed determined by pilot testing.

Design. Experiment 1 was a 2 x 2 between-subjects design with 22 participants per condition. The first factor consisted of animation vs. no animation. Participants who were assigned to the animation condition either watched or made predictions about the depth-first search animation; those in the no-animation condition worked with static paper materials. The second factor consisted of prediction vs. no prediction; participants in the prediction condition made a series of explicit predictions during training about the behavior of the depth-first search algorithm, those in the no-prediction condition did not.

2.2 Results

Our first concern was whether or not the participants, who were bright but not sophisticated with respect to computer algorithms, could understand even the basics of the depth-first search algorithm on the basis of the materials they had seen. Fortunately, they performed very well on the easy post-test questions, scoring an average of 7.58 out of 8, which is about 95% correct. There were also no group differences on the easy post-test questions, so participants most

likely grasped the fundamentals of the algorithm and could correctly search the post-test isomorphs of the graphs they had seen during training.

Next, we were interested in the effects of the animation on the predictions made during training by those participants in the prediction conditions. Participants in the prediction conditions had to make a total of 42 predictions (across two graphs) during the training phase. The average number of errors made by those who saw the animation was 0.46, while it was 1.19 for participants in the no-animation condition. This difference is statistically reliable ($F(1, 42) = 7.47$, $p < 0.01$) but, considering the low overall error rate, is not particularly large in a practical sense.

Finally, and most importantly, we were interested in the results of the difficult portion of the post-test, those problems that presented novel graphs to the participants. Each problem was scored as either being correct (the participant gave the complete ordered list of visited vertices correctly) or incorrect (the participant made one or more errors). There were seven such problems on the post-test, and the overall mean was 5.18 correct out of 7, or about 74% correct. The means (with standard deviations in parenthesis) are presented in Table 1.

While there appears to be an advantage for both animation and prediction on novel graph performance, neither main effect is statistically reliable, nor is there an interaction. The standard deviations are fairly high and both of the main effects might be considered marginal (Animation: $F(1, 83) = 3.82$, $MSE = 2.27$, $p = 0.054$; Prediction, $F(1,83) = 3.41$, $p = 0.068$). The interaction was non-significant, $F(1,83) = 0.19$, $p = 0.662$.

Table 1

Post-test Means for "Difficult" Problems (Experiment 1)

	Animation	No Animation	
Predict	5.73 (1.20)	5.24 (1.64)	5.49
No Predict	5.27 (1.72)	4.50 (1.40)	4.89
	5.50	4.86	

Note: Standard deviations are in parentheses.

One of the hypotheses we considered is that the advantage of animation is that it encourages prediction—this was certainly obvious with some of the participants who verbally made predictions while watching the animation even though they were not instructed to do so. If animations encourage prediction, then the appropriate test is a planned comparison in which the no-animation/no-prediction (or text-only) condition is compared to the other three conditions. This

difference is reliable, $F(1, 83) = 13.69$, $p = 0.016$, indicating that the participants in the no-animation/no-prediction condition were indeed outperformed by the others. There were no reliable differences among the other three groups.

2.3 Discussion

Clearly, participants in Experiment 1 learned the depth-first search algorithm reasonably well in all the conditions, answering a total of 85% of the post-test questions correctly. There was evidence for an advantage on the more difficult problems conferred by viewing the animation, though there was no evidence that the animation provided any advantage above and beyond that conferred by being required to make predictions. That is, Experiment 1 provides some support for the hypothesis that the ability to test predictions and get feedback is what confers any advantage associated with animation, since the participants who made predictions on paper did just as well as those who saw animations. However, there was some indirect evidence that those in the animation condition did acquire the knowledge faster, since they made fewer errors based on a fixed number of training trials.

Experiment 1 was limited in its ability to discriminate between the various possibilities for the results. First, the algorithm may have been too simple, limiting the ability of either the animation or the prediction task to provide much in the way of assistance to the learners. Second, the participants used in Experiment 1 were not the typical target population for algorithm animations as they were not computer science students. Perhaps computer science students would better comprehend animations because they have more practice working with various sorts of algorithms and visualizations. On the other hand, there may be less of an advantage for animations for this population since computer science students may be better at spontaneously generating their own visualizations. Third, the simplicity of the depth-first search algorithm involved may not play to the strengths of animation, such as the ability of the animations to make complex abstractions concrete. Finally, the post-test examined only one aspect of the students' understanding of the algorithm: their ability to execute the algorithm. Typical college-level courses on algorithms require knowledge about more than just how to execute an algorithm, such as the theoretical properties of the algorithm and the data structures used in the algorithm.

In order to address some of these concerns, we conducted a second experiment using advanced computer science students, a significantly more complex algorithm, and a more difficult post-test. It was hoped that these changes would allow us to detect possible advantages of animations in an ecologically valid situation.

3. Experiment 2

Experiment 2 was designed to address some of the concerns raised by Experiment 1 while still working within the same general experimental paradigm. We were again interested in the effects of animation and prediction and so we made use of the same basic design and procedures from Experiment 1, but with some key changes. First, we wanted to compare the results we found with relatively naive students and a simple algorithm to those based on a more sophisticated audience and a more complex algorithm. Thus, we tested upper-level computer science students on the use of a relatively esoteric data structure, the binomial heap. We hoped that this would address the issues surrounding the nature of the audience in Experiment 1 and the simplicity of the algorithm.

The second key difference between Experiments 1 and 2 was the structure of the post-test. With the naive students, we were concerned about whether or not they would retain the basics of the algorithm—which they clearly did. By using a more advanced group of students, we assumed that the simplest aspects of the algorithm would be easy for them so we could use the post-test to assess additional aspects of algorithm understanding. In Experiment 1, even the "difficult" post-test questions directly related to the strengths of the animation and the prediction task, since the test questions were concerned exclusively with the execution of the algorithm. Animations and predictions are directly tied to the step to step execution of an algorithm, and thus may be particularly useful in helping students learn how the algorithm executes on the micro level. However, the analysis of algorithms, and thus typical computer science courses on algorithms, are also concerned with the macro-level behavior of the algorithms, such as running time or memory use. Using advanced computer science majors allowed us to test these more analytical aspects of algorithm understanding along with performance on the step to step execution. We suspected that there might be a trade-off in that spending time working with the animation or prediction, while perhaps assisting learners to understand the execution of the algorithm, might take their focus off the analytic aspects and hurt their performance on post-test questions about these aspects.

3.1 Binomial Heaps

Binomial heap data structures are used as an implementation for an abstract data type called a priority queue. Priority queues are utilized in numerous computer science algorithms. They operate on nodes with key values (we can safely assume the key is an integer for simplicity). The most basic version of a priority queue involves two operations, insert and extract-minimum. Insert simply adds a new key-valued node to the priority queue, and extract-minimum removes and returns the node with least key value.

Note that a basic sorted list can be used to implement a priority queue. Under this scheme, the extract-minimum operation is fast and efficient because the smallest key is at the head of the

list. The insert operation can be very inefficient, however, because the entire list may need to be examined to find the proper insertion point. In general, computer scientists seek data structures without operations that are proportional to the entire size of the data structure. Binomial heaps are overall more efficient than using a simple list in this way.

A binomial heap is a collection (forest) of binomial trees (see Figure 2). A binomial tree is a heap-ordered tree data structure with smaller key values toward the root. Binomial trees always have a size that is a power of two. Larger binomial trees are made by joining two smaller trees of the same size. To join two binomial trees, their root values are compared. The root with higher key value is then linked in as a new child of the root with lower value, thus preserving the heap property.

The insert operation on a binomial heap simply adds a new binomial tree of size 1. If a tree of size 1 already exists in the heap, the two are joined to make a tree of size 2. This proceeds analogously until no more joining occurs.

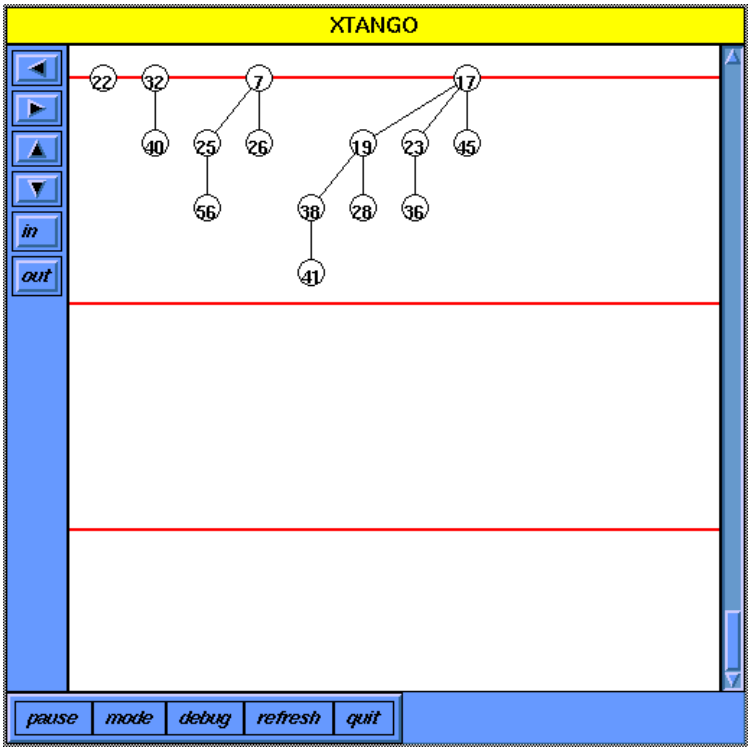
On an extract-minimum operation, the roots of all the binomial trees in the heap are searched and the smallest is found. It is removed and all of its children are elevated to be trees in the binomial heap. This may introduce trees of like sizes, so joining may need to be done. Joining begins by checking for trees of size 1 and proceeds upwards. Both the joining and extract-minimum operations may require examining the root of all the trees in the binomial heap. But because of the unique way that trees are combined into power of two sizes, only a logarithm of the total size of the heap number of steps is involved.

For more details on binomial heaps and their operations, consult any comprehensive computer science algorithms text such as (Cormen et al., 1990).

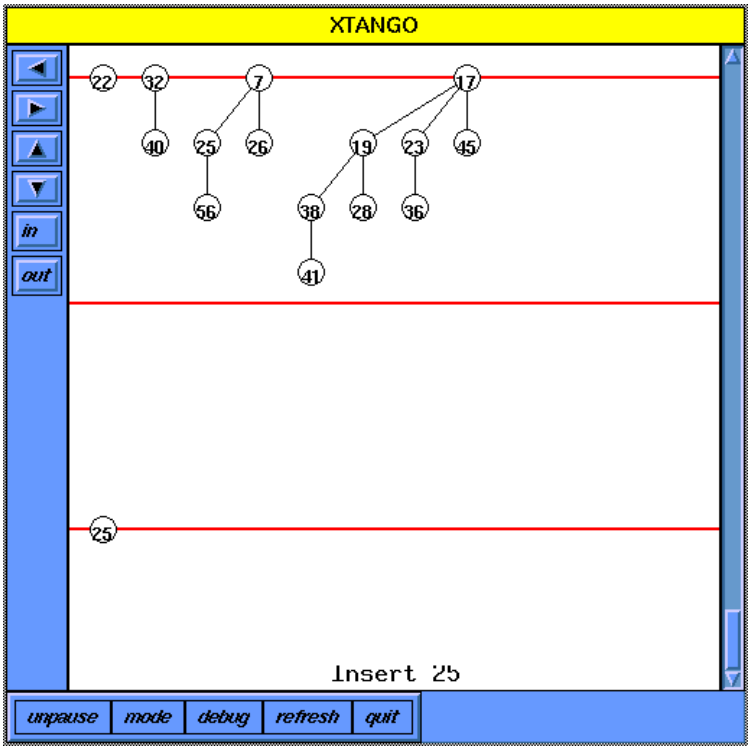
The animation of the binomial heap data structure and algorithm was built using the XTango algorithm animation system (Stasko, 1992) and utilizes the representation of a binomial heap common in algorithms textbooks. The animation uses three work areas, each of which is indicated by a horizontal red line as illustrated in Figure 2a. The top work area is where stable binomial heaps, that is, heaps after operations, are drawn. All the binomial trees comprising the binomial heap are drawn in increasing tree size from left to right. The lowest work area is where nodes from a new insert operation are initially placed and where "new" trees from an extract-minimum operation (i.e., the children of the root) are originally placed. The middle work area represents an intermediate state. Trees reside here in the midst of an insert or extract-minimum operation; trees of the same size are joined, linked, and made into larger trees. All of these linking operations are illustrated using smooth, gradual animations that allow the viewer to track the context (e.g., heap location, node position) of the operation. A label at the bottom of the animation window lists the name of the operation currently taking place.

Figure 2: Screen Shots of Animation for Binomial Heap (BH)

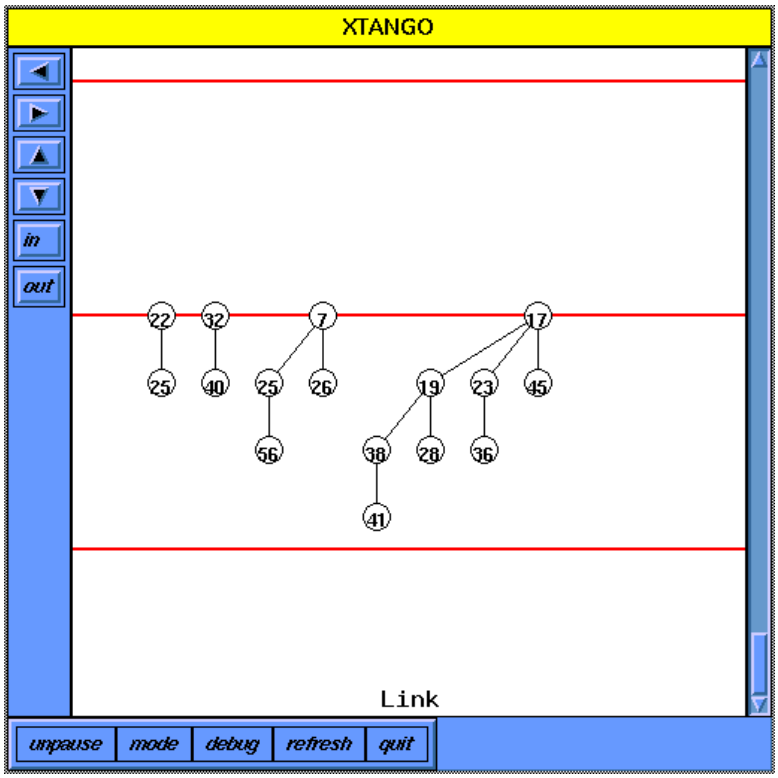
a)



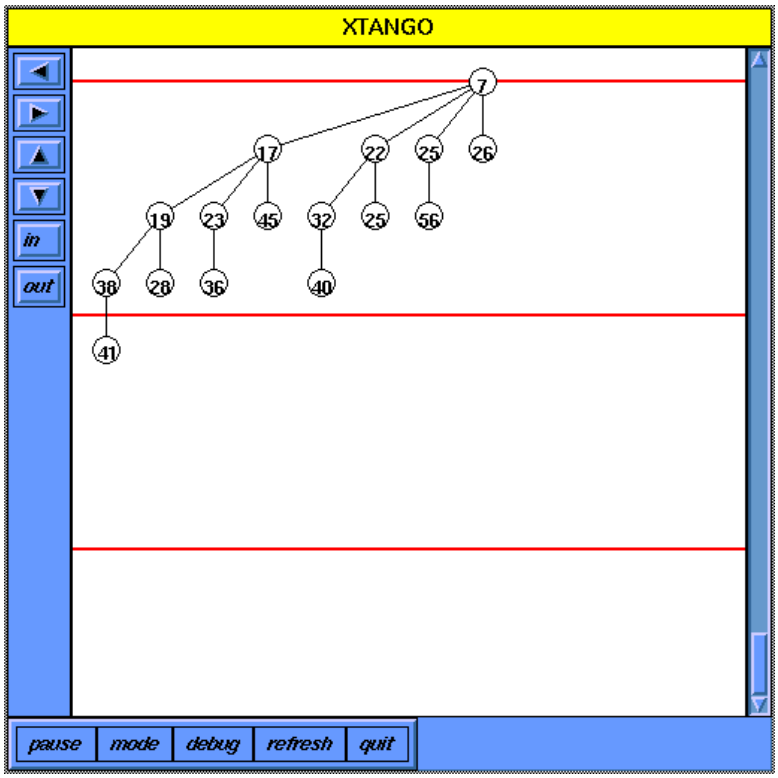
b)



c)


XTANGO
in
out
22
32
7
17
25
40
25
26
19
23
45
56
38
28
36
41
Link
unpause
mode
debug
refresh
quit

d)


XTANGO
in
out
7
17
22
25
26
19
23
45
32
25
56
38
28
36
40
41
pause
mode
debug
refresh
quit

As described above, the binomial heap algorithm that we examined involves two operations: insert and extract-minimum. During an insert operation, the new tree appears at the lowest work area (Figure 2b) and is then moved to the middle work area. Next, all of the binomial trees in the heap above are brought down to the middle work area and the appropriate union and link operations are illustrated (Figure 2c). Finally, the resulting trees are elevated to the top work area again (Figure 2d).

During an extract-minimum operation, the smallest key-valued node in the binomial heap (one of the roots at the top) is flashed in red and removed. All of the resulting trees (child subtrees of the root) are moved to the lowest work area. At this point, all the trees at the lower work area and at the top work area are moved into the middle work level. The trees are moved in increasing size so that the middle work area contains all the intermediate trees in growing size from left to right. The linking operations then commence and the resulting binomial trees move up to the top stable work area.

3.2 Method

Participants. The participants in Experiment 2 were 62 undergraduate computer science majors (typically juniors and seniors) at the Georgia Institute of Technology who participated for extra credit in an upper-level computer science course. Participants were screened to ensure they had no previous experience with binomial heaps, though they were required to have some background in algorithm analysis. The participants were again very high-ability students with a mean self-reported SAT-Quantitative score of 672 and SAT-Verbal score of 567.

Materials and Procedure. The procedure was similar to that in Experiment 1. Participants first filled out the consent form and background questionnaire then watched a videotape of an 11-minute lecture on binomial heaps. After watching the videotape, participants were given an 12-page text describing binomial heaps which was also adapted from the chapter in Cormen et al. (1990) covering binomial heaps. Participants in the no-animation/no-prediction condition were given 35 minutes to read the text, all others were given 20 minutes. Those who received only 20 minutes with the text then either simply watched the algorithm animation (animation/no-prediction condition), made explicit predictions while watching the animation (animation/prediction condition), or made explicit predictions from printed graphs (no-animation/prediction condition). All participants then received the post-test, on which they had 25 minutes to work.[2]

[2]Note that in Experiment 1 participants had an unlimited amount of time to work on the post-test. However, given the relatively long time we felt participants would require to do Experiment 2, we decided to limit how much time could be spent on the post-test.

Participants in the prediction condition had to predict the state of a binomial heap after a specified operation was performed. That is, participants were shown a binomial heap and asked to draw the binomial heap that would be the result of an insert or an extract-min operation. Those watching the animation were shown the correct answer by the animation after making each prediction, while those making predictions on paper were shown a static picture of the correct heap.

The number of errors during prediction was again recorded for those participants who made explicit predictions, and again there was a post-test. Post-test questions could be divided loosely into two categories, procedural questions and conceptual questions. Procedural questions focused on the execution of the algorithm and required participants to perform operations on binomial heaps. Most conceptual questions, on the other hand, required knowledge about the abstract properties of binomial heaps. Questions on the post-test were again scored stringently as being either completely correct or incorrect.

The apparatus was identical to what was used in Experiment 1.

Design. The design was identical to Experiment 1's, a between-subjects 2 x 2 design with animation vs. no animation and prediction vs. no prediction as the factors. There were 16 participants in each of the two no-animation groups and 15 in each of the two animation groups.

3.3 Results

One of the results found in Experiment 1 was an advantage for those seeing the animation during the prediction training phase of the experiment. This result was not replicated. Participants who saw the animation averaged 2.77 errors (out of a possible 12) and those who did not averaged 2.54 errors. This difference is not statistically reliable and does not even match the direction of the difference found in Experiment 1. However, the number of errors in the prediction phase was correlated with performance on the procedural post-test questions, $r(30) = 0.70$, $p < 0.01$.

We did not expect to find an advantage for the animation or prediction conditions with respect to performance on the conceptual questions. If anything, the no-animation/no-prediction condition might yield the best performance here because this material was covered primarily in the text and participants in this group had more time to study the text. No differences were found among any of the groups. Mean number correct out of a possible 17 are shown in Table 2, with standard deviations in parentheses. Performance on the conceptual questions was also correlated with performance on the procedural questions, $r(61) = 0.51$, $p < 0.01$.

On the other hand, we expected to see some kind of separation between the various groups on the procedural questions. Mean number correct out of a possible seven are shown in Table 3. The relative ordering of the means mirrors the results of Experiment 1, though the differences are not as robust as the differences found in Experiment 1. This might be due partly to the relatively

high variability in the conditions. The main effects of animation and prediction were not reliable, and the planned comparison between the no-animation/no-prediction group and all the others is only marginally reliable ($F(1, 58) = 3.70$, $p = 0.059$).[3]

Participants were correct on 63% of the post-test questions. There was some evidence that participants in Experiment 2 could be best characterized as belonging to one of two groups, those who did well and those who did poorly. Participants were divided into those who fell above or below the median score on the post-test. This median split predicts 64% of the variance in total post-test scores ($r(61) = 0.80$, $p < 0.01$), which was far and away the best predictor of overall performance. There was no reliable association between the condition a participant was assigned to and whether or not they were a strong performer or a weak performer.

Table 2

Post-test Means for Conceptual Questions (Experiment 2)

	Animation	No Animation	
Predict	9.93 (3.17)	10.94 (3.11)	10.45
No Predict	10.80 (3.08)	9.94 (3.11)	10.35
	10.37	10.44	

Note: Standard deviations are in parentheses.

Table 3

Post-test Means for Procedural Questions (Experiment 2)

	Animation	No Animation	
Predict	5.13 (1.30)	4.94 (2.05)	5.03
No Predict	5.00 (1.60)	4.00 (2.19)	4.48
	5.07	4.47	

Note: Standard deviations are in parentheses.

[3]Our power to detect a "large" effect was very good, 87% (Cohen, 1988). Thus, if there really was a large effect, we should have found it. Obviously, the power would be lower if the effect size were smaller (71% for a medium-large effect), but if the effect is a small one it is not clear that building an animation would be worth the trouble.

3.4 Discussion

Interpreting the results of Experiment 2 is difficult. We were hoping that since we made the materials more difficult, and therefore provided more of an opportunity for animation to benefit learners, Experiment 2 would provide a more sensitive test than did Experiment 1. While we did find some evidence for benefits of animation and prediction similar to those found in Experiment 1, this evidence is weakened by the generally high variability in performance. It may be the case that the animation and predictions were useful only to those participants who already had a good understanding of the algorithm by the time they reached the post-test stage of the experiment. The animations and predictions may simply not have been comprehensible to those participants who were still struggling with the basics of the algorithm. Some aspects of the data suggest this, as one of the more common errors found in the answers to the procedural questions among the weak performers were the creation of data structures that did not even meet the definition of a binomial heap. This suggests that they did not fully understand the foundational material presented in the videotape and the text. Thus, it is not clear that viewing the animation or making predictions provides any benefit if the learner has not acquired some fundamentals.

On the other hand, it may be that the animation and prediction activities were not useful to the high-performers since they may have understood the material well enough to solve the post-test problems before seeing the animation or making predictions. The fact that performance on the prediction task was strongly correlated with performance on the procedural part of the post-test supports this claim. That is, participants who made a lot of errors in the prediction task did not seem to learn enough from this experience to improve their later performance. In general, good performers did well in every phase of evaluation and poor performers did poorly on all phases.

Given the possibility that weak learners and strong learners will not be much affected by the animation and prediction manipulations, a future experiment could replicate Experiment 2, but include assessments after the video/text portion of the experiment and then after the animation/prediction portion and compare the changes (if any). This within-subject approach might be more sensitive to effects of animation/prediction (although the inclusion of the first post-test might affect performance on the second post-test as well as cuing learners in the animation/no-prediction condition to make predictions more frequently than they might otherwise).

4. General Discussion

The results from the two experiments show an unreliable benefit of animations and predictions on students' ability to solve procedural and conceptual problems about algorithms. This weak effect of animations is inconsistent with the intuitions of many algorithm teachers. It seems clear that the sheer use of algorithm animation does not automatically enhance learning--

there are a host of serious unresolved issues to consider. Claims about the effectiveness of algorithm animations for learning should be treated with skepticism if the animations are not accompanied by empirical demonstrations of their effectiveness. In addition, it is clear that an algorithm animation must be carefully designed and crafted; just putting one together and showing it to the learner is not sufficient. A careful task analysis must be done to show what knowledge the animation will, in theory, be helping the learner to acquire.

One possible explanation for the lack of an animation effect in the present study is that prior knowledge of learners, particularly in Experiment 2, might have interacted with the animations (and predictions) in unanticipated ways. That is, it is not obvious how the information conveyed in the animation or provided by making predictions interacts with the knowledge that the learners bring to bear when they initially engage in these activities. Animations and predictions with complex algorithms may help only some learners, those with just exactly the right amount of knowledge to make use of the information provided, but not so much so that the information provided is redundant with what they already know.

A related question is whether the animations we used were the right ones to help learners understand the algorithms. We drew on our prior experiences with algorithm animation as we made our choices in constructing the animations. We designed what we thought were "good" animations, but perhaps we made some unsatisfactory decisions. Should an animation reflect a "novice" or "expert" point of view for how the algorithm operates? Should learners be allowed to choose the inputs into the algorithm? Should the animation be veridical with respect to the animation or should it gloss over certain details in order to make the big picture clear? Should the animation be annotated with additional features such as pseudocode, or it should it be relatively unadorned? There are, as yet, no clear guidelines for the construction of algorithm animations, not just from a psychological perspective but from an implementational one. What visual elements should be included? How should they look? What should the flow and sequencing be? These sorts of questions can motivate additional studies into the potential benefits of animations on algorithm learning and understanding.

Another important issue to consider is what kinds of learning animations might help the most. For instance, perhaps animations are best able to help learners acquire the "big picture" concerning the algorithm rather than the small details. However, there is the paradoxical problem that an animation that shows the big picture or emergent qualities might be appreciated only by those that already understand the algorithm at the mechanical level!

Another possibility is that animations might not aid overall learning or performance, but may aid how *quickly* a student learns. We made sure that students in the different training conditions spent roughly the same amount of time with the training materials, but perhaps that control wiped out the benefit that an animation might provide. There is some evidence from

ethnographically-motivated work that learners do make use of animations in the learning process (Kehoe & Stasko, 1996). An idea to consider for future research is to allow learners in the different conditions to spend as little or as much time with the different materials as they wish and then to examine performance.

In this same vein, we may not have developed the most sensitive tests for assessing what students learned from the training phase. For instance, tasks that require students to make rapid responses might demonstrate that those who studied animations are able to run their mental models of the algorithm more quickly than students who did not see the animation. Test tasks that require students to use the algorithm more creatively, such as using the algorithm to help comprehend a related algorithm, also might show benefits from watching animations.

The various issues raised above suggest that additional research on algorithm animation can benefit from a careful task analysis. That is, one must decide what it is a student should learn about an algorithm and how this information could potentially be conveyed. Perhaps some of the information can best be conveyed through animations while other information is best taught through other methods. Simply put, algorithm animations may not be helpful in certain circumstances. They may provide incremental assistance, but it remains to be shown that the benefits cannot be achieved with alternate instructional aids such as basic drill (as found in Experiment 1). Constructing algorithm animations generally requires serious programming effort, and it still has to be demonstrated that the benefits justify this cost. If the same pedagogical advantages can be realized with less labor-intensive materials, then the less labor-intensive methods make more sense. It is incumbent upon educators and animation-builders to carefully examine their assumptions about what students will learn from an animation, and why it is that an animation is necessary to convey the desired information.

Nevertheless, the data from this study coupled with that of other prior formal empirical studies does show small, unreliable benefits for the animations. At least these results are in the right direction. We still believe that algorithm animations can assist the instructional process in valuable ways, particularly if task analyses are used in order to take advantage of the hypothesized strengths of animations. Therefore, we plan to further investigate the potential effects of algorithm animations and why those effects occur.

References

Anderson, J.R. (1993). *Rules of the mind.* Hillsdale, NJ: Erlbaum.

Baecker, R., & Small, I. (1990). Animation at the interface. In B. Laurel (Ed.), *The art of human-computer interface design.* Reading, MA: Addison-Wesley (251-267).

Brown, M.H. (1988a). Perspectives on algorithm animation. In *Proceedings of the ACM SIGCHI '88 Conference on Human Factors in Computing Systems*, 33-38.

Brown, M.H. (1988b). Exploring algorithms using Balsa-II. *Computer*, 21 (5), 14-36.

Brown, M.H. (1991). ZEUS: A system for algorithm animation and multi-view editing. In *Proceedings of the 1991 IEEE Workshop on Visual Languages*, 4-9.

Catrambone, R. (1996). Generalizing solution procedures learned from examples. *Journal of Experimental Psychology: Learning, Memory, and Cognition,* 22(4), 1020-1031.

Chi, M.T.H., Bassok, M., Lewis, R., Reimann, P., & Glaser, R. (1989). Self-explanations: How students study and use examples in learning to solve problems. *Cognitive Science*, 13, 145-182.

Chi, M.T.H., De Leeuw, N., Mei-Hung, C., & Levancher, C. (1994). Eliciting self explanations. *Cognitive Science*, 18, 439-477.

Cohen, J. (1988). Statistical power analysis for the behavioral sciences (2nd ed.). Hillsdale, NJ: Erlbaum.

Cormen, T. H., Leiserson, C. E., & Rivest, R. L. (1990). *Introduction to algorithms*. Boston, MA: MIT Press.

Gloor, P.A. (1992). AACE--Algorithm animation for computer science education. In *Proceedings of the 1992 IEEE Workshop on Visual Languages*, 25-31.

Kehoe, C. M., & Stasko, J. T. (1996). *Using animations to learn about algorithms: An ethnographic case study*. Unpublished manuscript.

Lawrence, A.W., Badre, A.M., & Stasko, J.T. (1994). Empirically evaluating the use of animations to teach algorithms. In *Proceedings of the 1994 IEEE Symposium on Visual Languages*, 48-54.

Mayer, R.E., & Anderson, R.B. (1991). Animations need narrations: An experimental test of a dual-coding hypothesis *Journal of Educational Psychology*, 83 (4), 484-490.

Mayer, R.E., & Anderson, R.B. (1992). The instructive animation: Helping students build connections between words and pictures in multimedia learning. *Journal of Educational Psychology*, 84 (4), 444-452.

Palmiter, S. (1993). The effectiveness of animated demonstrations for computer-based tasks: A summary, model, and future research. *Journal of Visual Languages and Computing*, 4 (1), 71-89.

Price, B.A., Baecker, R.M., & Small, I.S. (1993). A principled taxonomy of software visualization. *Journal of Visual Languages and Computing*, 4 (3), 211-266.

Rieber, L.P. (1990). Effects of animated graphics on student learning. *Journal of Educational Psychology*, 1, 123-321.

Rieber, L.P., Boyce, M.J., & Assad, C. (1990). The effects of computer animation on adult learning and retrieval tasks. *Journal of Computer-Based Instruction*, 17 (2), 46-52.

Robertson, G.G., Card, S.K., & Mackinlay, J.D. (1993). Information visualization using 3D interactive animation. *Communications of the ACM*, 36 (4), 57-71.

Stasko, J.T. (1992). Animating algorithms with XTANGO. *SIGACT News*, 23 (2), 67-71.

Stasko, J.T. (1990). TANGO: A framework and system for algorithm animation. *Computer*, 23 (9), 27-39.

Stasko, J.T., Badre, A., & Lewis, C. (1993). Do algorithm animations assist learning? An empirical study and analysis. In *Proceedings of the INTERCHI '93 Conference on Human Factors in Computing Systems*, 61-66.

Stasko, J.T., & Kraemer, E. (1993). A methodology for building application-specific visualizations of parallel programs. *Journal of Parallel and Distributed Computing*, 18 (2), 258-264.

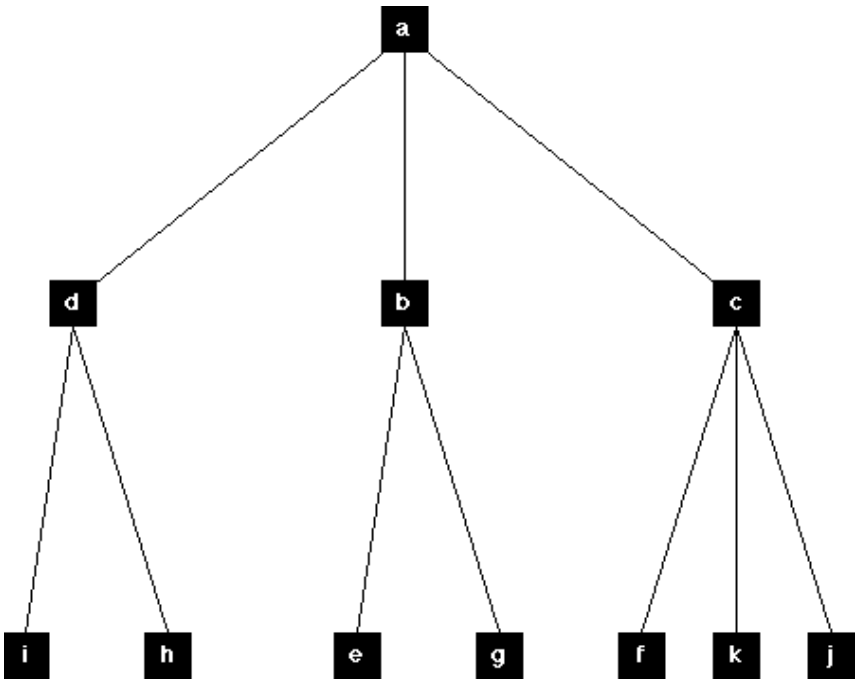
Sukaviriya, P. (1990). Coupling a UI framework with automatic generation of context-sensitive animated help. In *Proceedings of the '90 ACM SIGGRAPH Symposium on User Interface Software and Technology*, 152-166.

Appendix A

Sample of Prediction Task Materials for No-Animation Condition

in Depth-First Search Experiment (Experiment 1)

(Excerpt from instructions read to the participant): You will see a graph and I will ask you at each stage where the depth first search will go next. Take your time and try to answer the questions correctly.

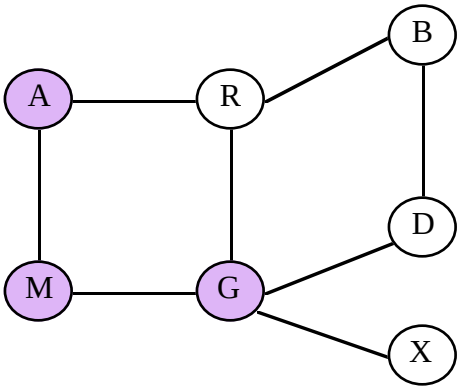


Appendix B

Sample of Post-Test Questions for Depth-First Search (Experiment 1)

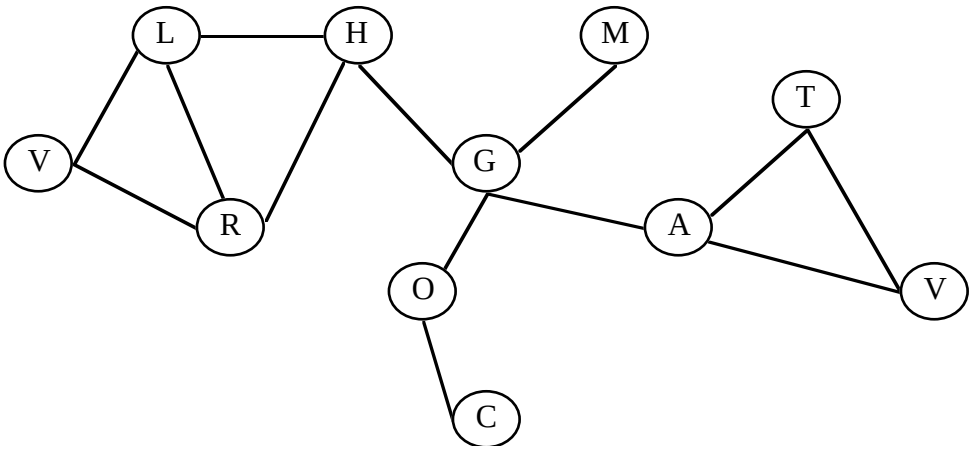
"Easy" Question

In the following graph, assume that all darkened vertices have already been visited in a depth first search. Also assume that we have just visited vertex G. What would be the next vertex that we visit?



"Difficult" Question

Starting with vertex A, list the order of vertices as they are encountered in a depth first search of the graph below. Also list the order in which vertices are **first** encountered.

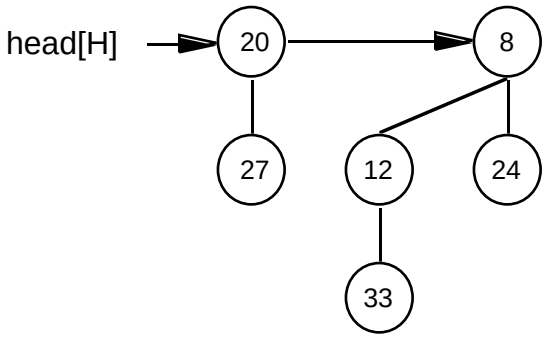


Appendix C

Sample of Prediction Task Materials for No-Animation Condition

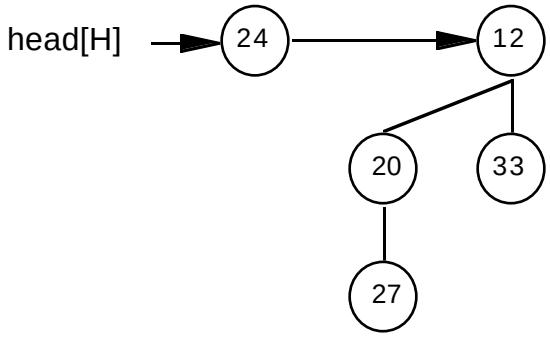
in Binomial Heap Experiment (Experiment 2)

Draw the results of doing an EXTRACT-MIN on the binomial heap below.



(the text and figure below would appear on the page following the above text and figure)

The correct answer to the previous problem is the binomial heap shown below.

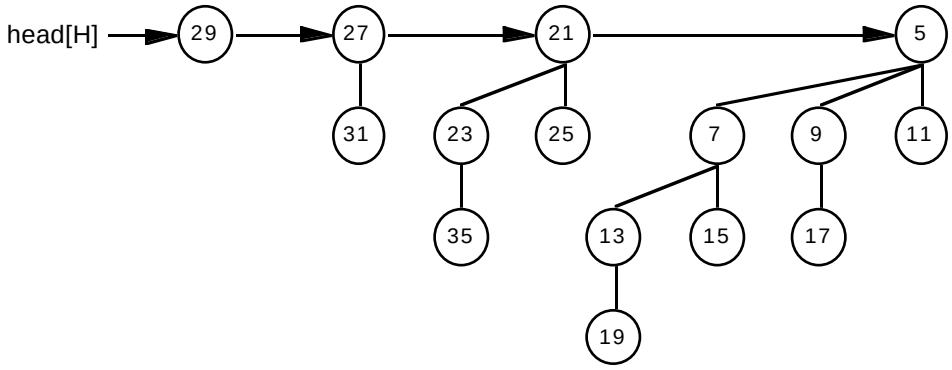


Appendix D

Sample of Post-Test Questions for Binomial Heap (Experiment 2)

Procedural Question

Suppose "33" were inserted into the heap pictured below. Please draw the result.



Conceptual Question

What is the worst case running time for INSERT on a binomial heap?

Author Notes

This research was supported by a grant from the Graphics, Visualization, and Usability Laboratory of the Georgia Institute of Technology. Parts of Experiments 1 and 2 were reported at the 7th Annual Winter Text Conference, Jackson Hole, January 1996 and at the Basic Research Symposium at the CHI '96 Conference, Vancouver, April 1996.

Correspondence concerning this article should be addressed to Richard Catrambone, School of Psychology, Georgia Institute of Technology, Atlanta, Georgia 30332-0170. Electronic mail may be sent to rc7@prism.gatech.edu.